

**ReactJS.**

Thinking in React

# Developing a React web app

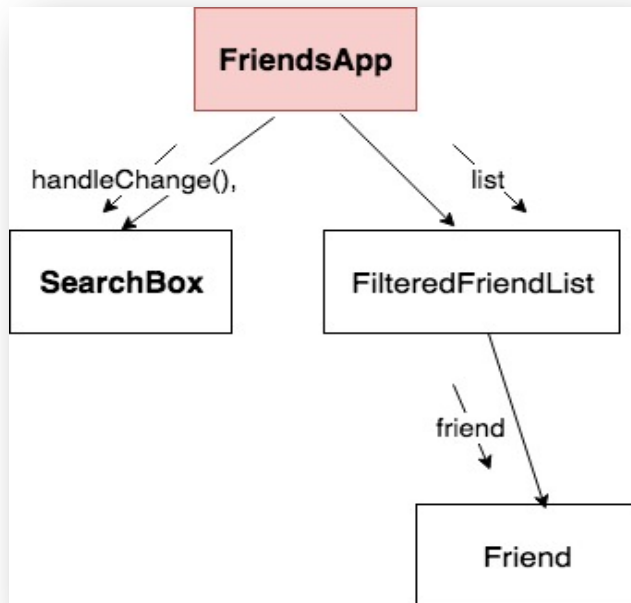
- **Step 1: Break the UI into a component hierarchy.**
- **Step 2: Build a static version of the app.**
- **Step 3: Identify the minimal representation of UI state.**
- **Step 4: Identify where your state should live.**
- **Step 5: Add inverse data flow, if required.**

# Starting point.

- **At the start of the development process we have:**
  - 1. A mock-up of the UI.**
  - 2. (Optionally) A JSON representation of the web API data model.**

# Step 1: Break the UI into a component hierarchy.

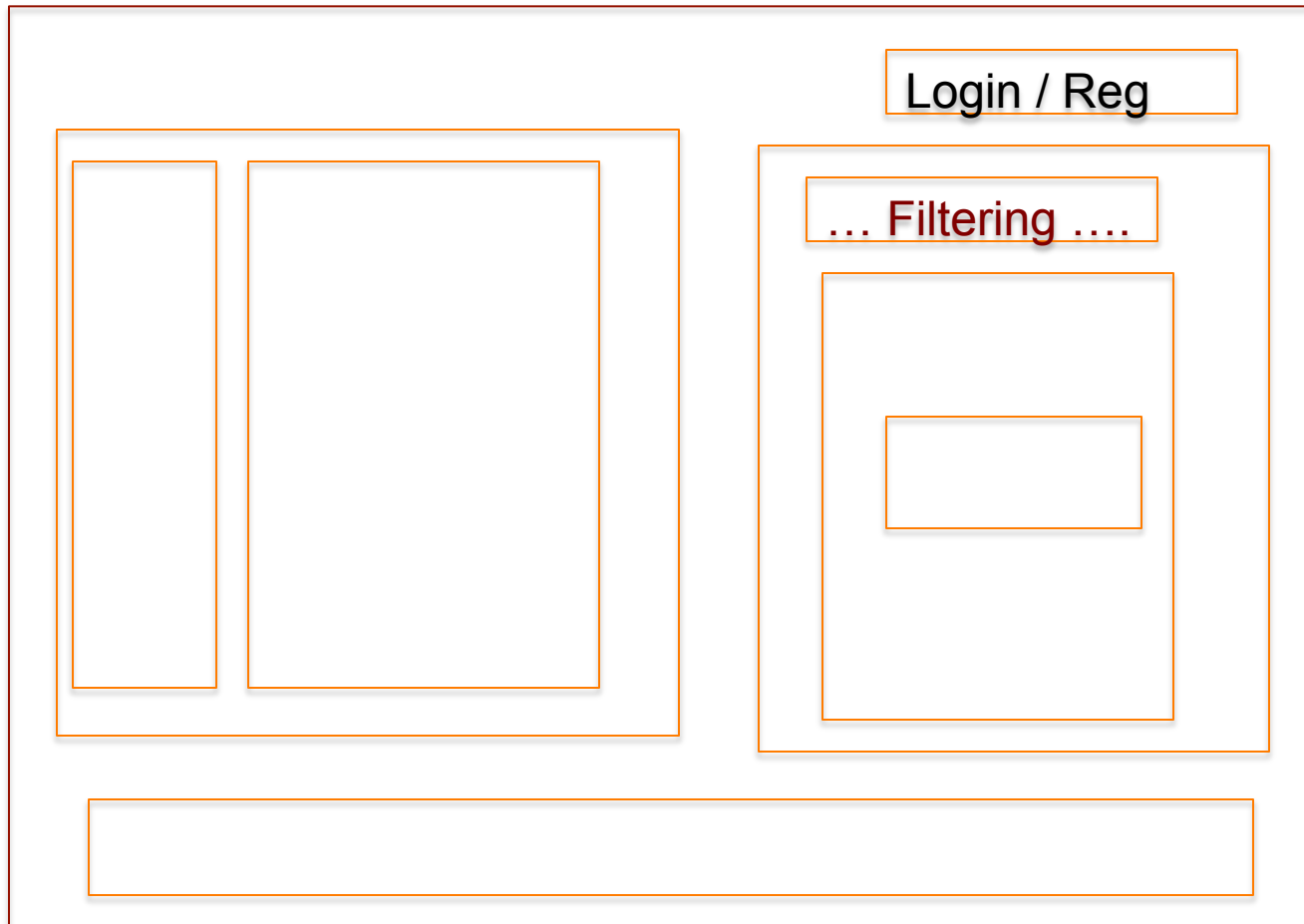
- Possibly use the data model as a guide.
  - The UI and data models tend to adhere to the same information architecture.



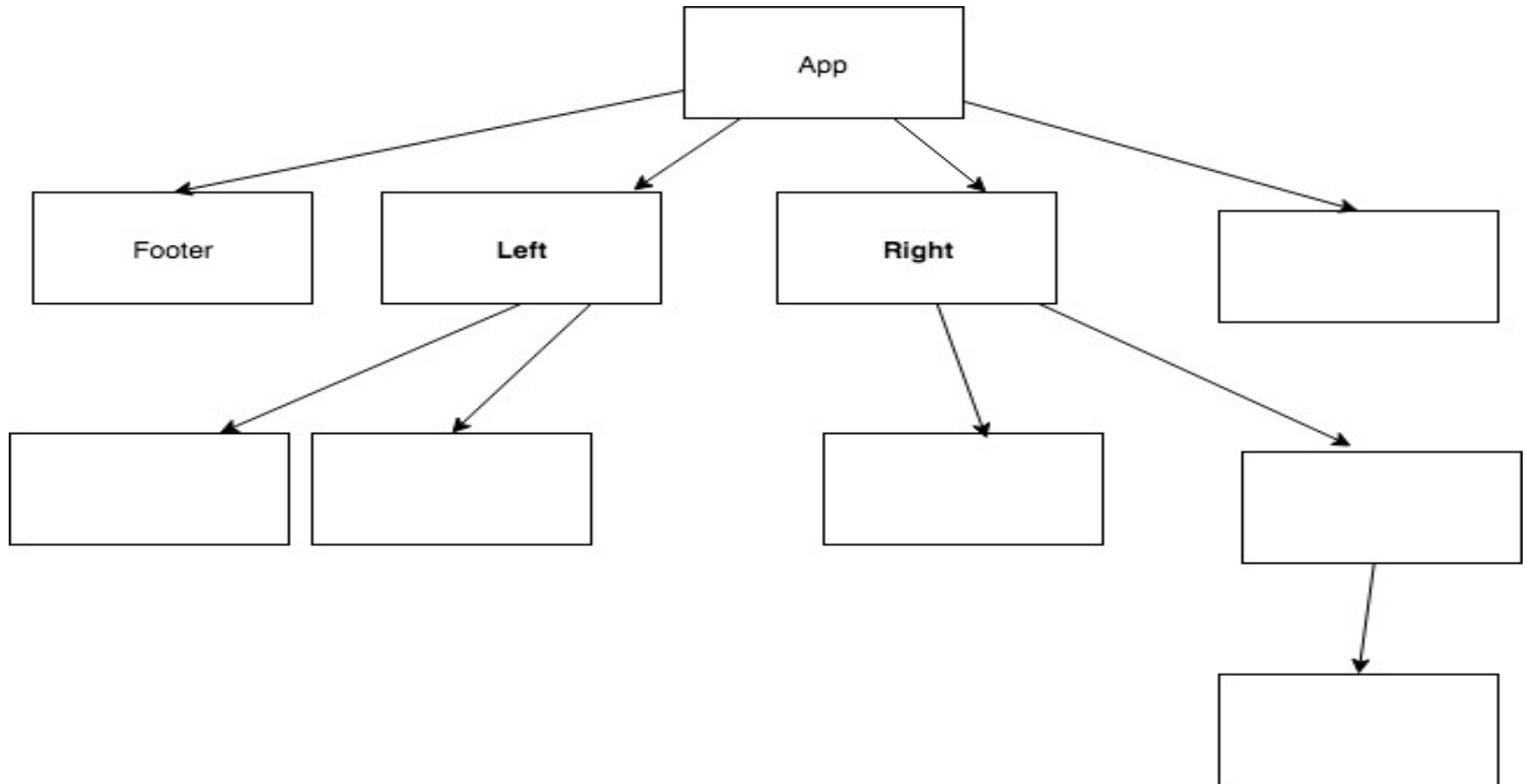
## Friends List

- **Joe Bloggs**  
[jbloggs@here.con](mailto:jbloggs@here.con)
- **Paula Smith**  
[psmith@here.con](mailto:psmith@here.con)
- **Catherine Dwyer**  
[cdwyer@here.con](mailto:cdwyer@here.con)
- **Paul Briggs**  
[pbriggs@here.con](mailto:pbriggs@here.con)

# Component hierarchy - Abstract example



# Component hierarchy - Abstract example



## Contact List 5

Add Contact

Contact 1

Contact 1  
123 Test St  
132-3212

EditDelete

Contact 2

Contact 2  
23 Main St  
934-4329

EditDelete

Contact 3

Contact 3  
4 Lower St  
432-5832

EditDelete

Contact 4

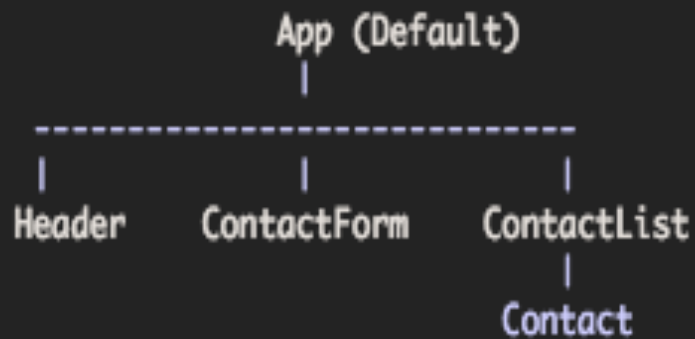
Contact 4  
49 Upper Street  
934-4290

EditDelete

Contact 5

Contact 5  
4 High Street  
933-3390

EditDelete



```
contacts : [
  {
    "name": "Contact 1",
    "address": "123 Test St",
    "phone_number": "132-3212"
  },
  {
    "name": "Contact 2",
    "address": "943 Main Road",
    "phone_number": "934-4329"
  },
  { },
  { }
],
]
```

# Hacker News

14 The button that is not. Comments

12 Google Nears \$1B Investment in SpaceX Comments

11 India - Tiger population sees 30% increase. Comments

2 Coinbase Raises \$75M from DFJ Growth, USAA, and More Comments

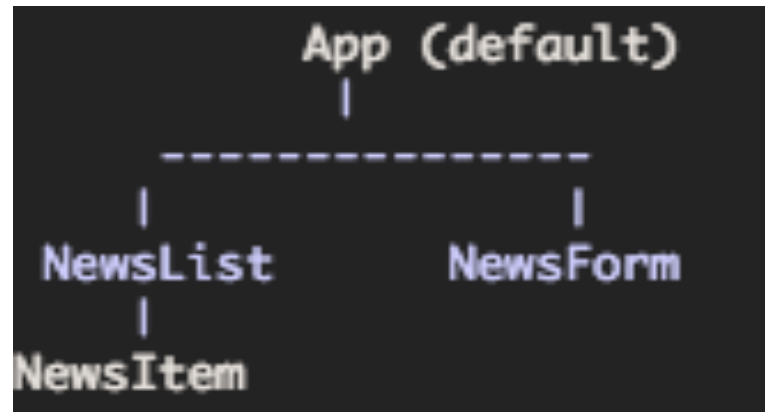
## Add a new post

Title

Link

Post

- Component hierarchy:





# Step 1: Break the UI into a component hierarchy.

- **Additional criteria for devising component breakdown:**
  1. **The single responsibility principle.**
  2. **If it's doing too much, break it up.**
  3. **If Component has too much code, break it up**
- **[ Same principles when dealing with Object Oriented design.]**

## Stap 2: Build a static version.

- **Using a sample data set, render the UI but ignore all interactivity.**
- **All components will only have a render method; no lifecycle methods or event handlers or state, yet.**
- Design Principle: **Decouple structure from interactivity, initially.**
- **“Lots of typing but little thinking.”**
- **Use Storybook to build components.**
  - **Helps determine component prop requirements. \*\*\*\***
  - **Start with ‘leaf’ components, and work up the hierarchy, e.g. Phone → PhoneList**
  - **Consider multiple stories for a component, e.g. prop boundry values, default value.**

## Step 3: Identify the minimal (but complete) representation of UI state.

- Try to keep as many of the components as possible stateless.
  - Stateless components simply render props data
- Follow the DRY principle (Don't Repeat Yourself):
  - Identify the absolute minimal representation of the state the app needs and compute everything else on-demand, e.g. list of matching friends,
- Common app pattern – A stateful component computes the props for its subordinates based on current state, domain data and/or its own props.

## Step 3: Identify UI state.

- **What shouldn't go in State?**
  1. **Domain Data** - data retrieved from a Web API/service.
    - Temporarily stored on the client-side, but not as UI state.
  2. **Computed data**, e.g. subset of matching friends
    - Avoids keeping computed state in sync with user interaction; Just re-compute it when necessary.
  3. **Copies of props**: Props are the 'source of truth'.
    - Unless props' previous value(s) effects rendering.
  4. **React components**; State should always be JSON-serializable.

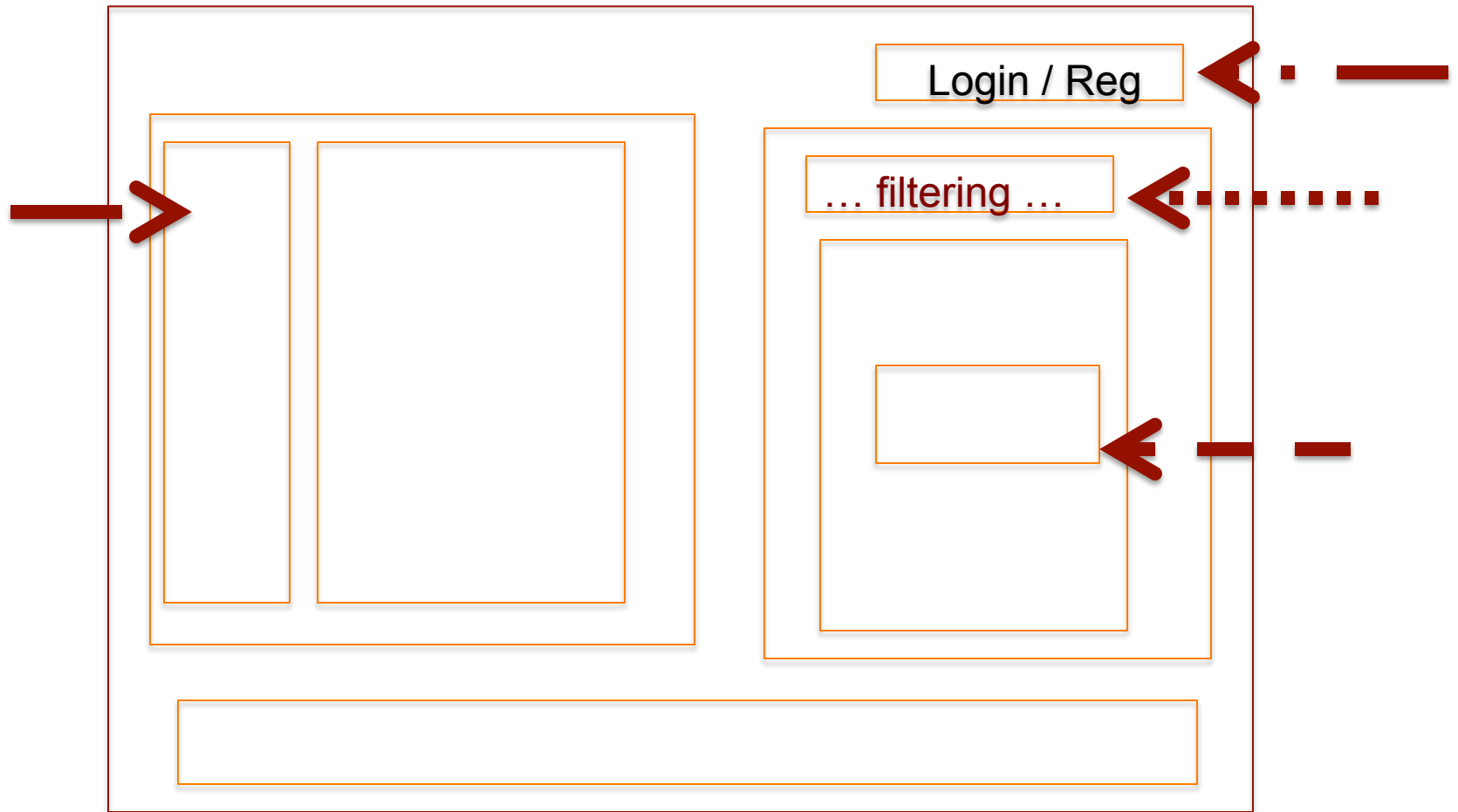
## Step 3: Identify UI state.

- **What should go in state?**
  - **Data that an event handlers changes, e.g. counter**
  - **User dialogues selections – check box, menu, radio button. input text fields.**
- **How to identify state:**
  - 1. Identify all of the places where data appears in the UI.**
  - 2. For each one, ask a set of questions:**
    - I. Is it passed in via props? If so, probably isn't state.**
    - II. Is it modifiable? If not, probably isn't state.**
    - III. Can you compute it based on any other state or props? If so, it's not state.**

# Example: Filtered Friends app

- **Think of all of the places where data appears in the UI:**
  1. **Full List of friends.**
    - **Supplied from web API → Not state.**
  2. **Search text.**
    - **Modifiable, User input → State.**
  3. **Filtered list of friends.**
    - **Computed → Not State.**
  4. **Friend details (name, etc)**
    - **Passed in as props, Not modifiable → Not state**

# Identify UI state - Abstract example

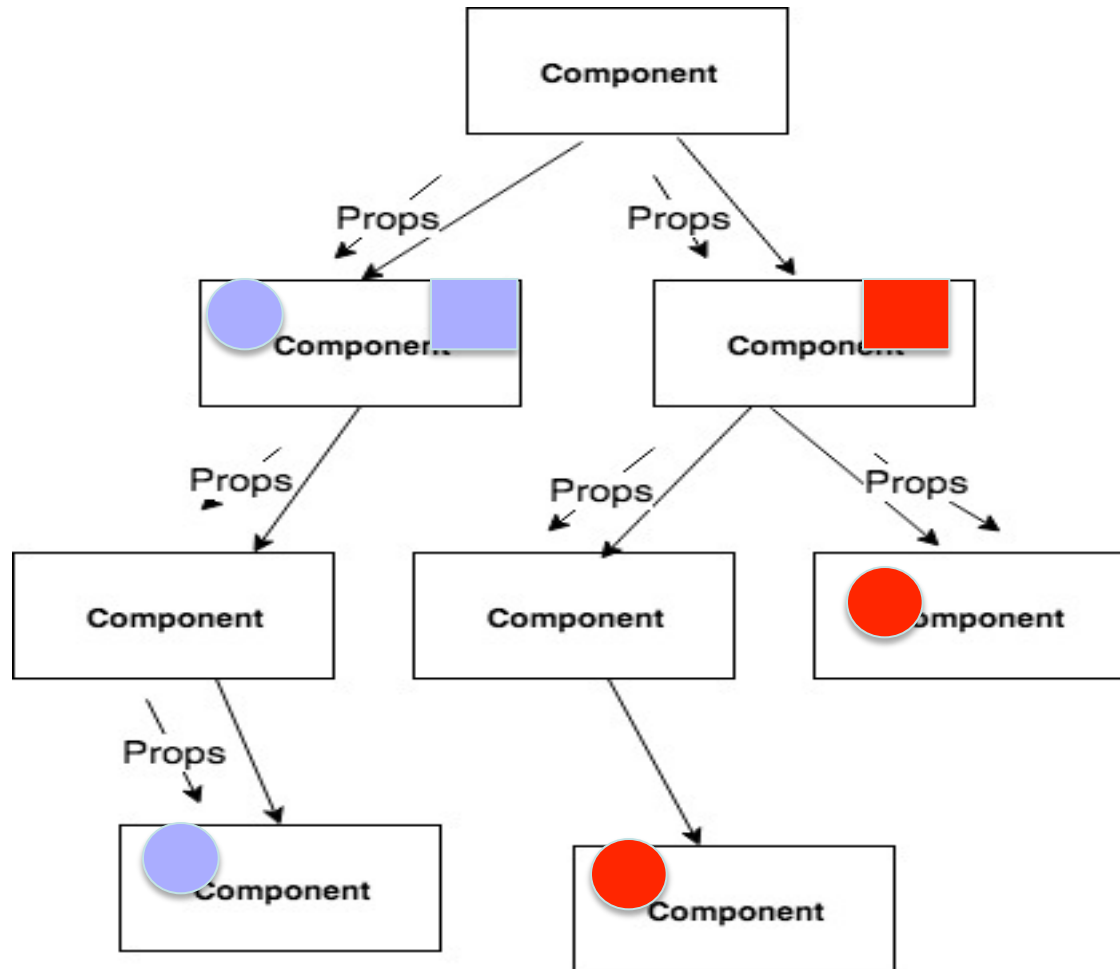


## Step 4: Identify where state should live.

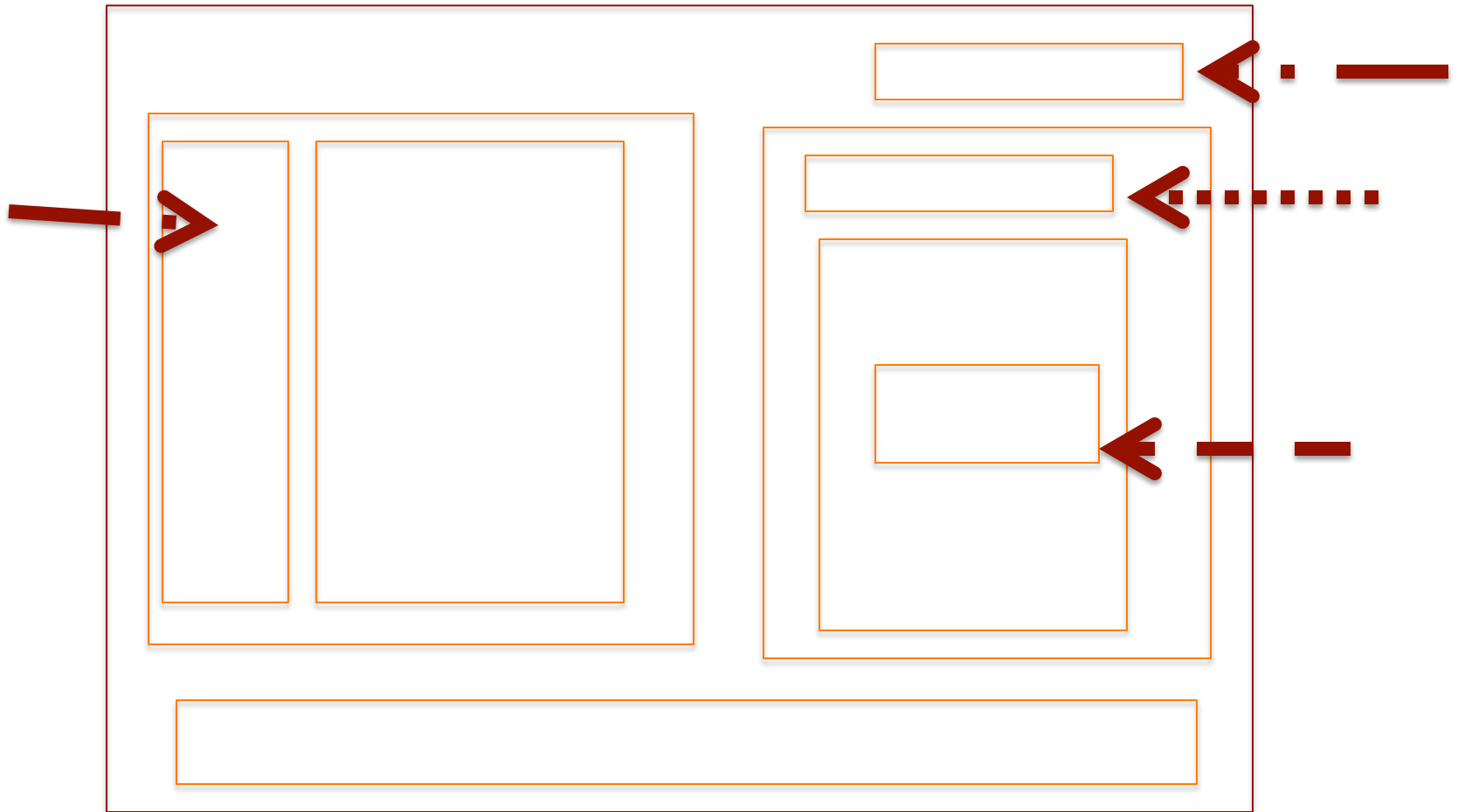
- **For each piece of UI state, go through this process:**
  - 1. Identify every component that renders something based on its value.**
  - 2. From 1 above, identify the ‘common’ ancestor component.**
  - 3. If there is no obvious candidate, create a new ancestor component.**
- **Add state initialization code to selected ancestor component**



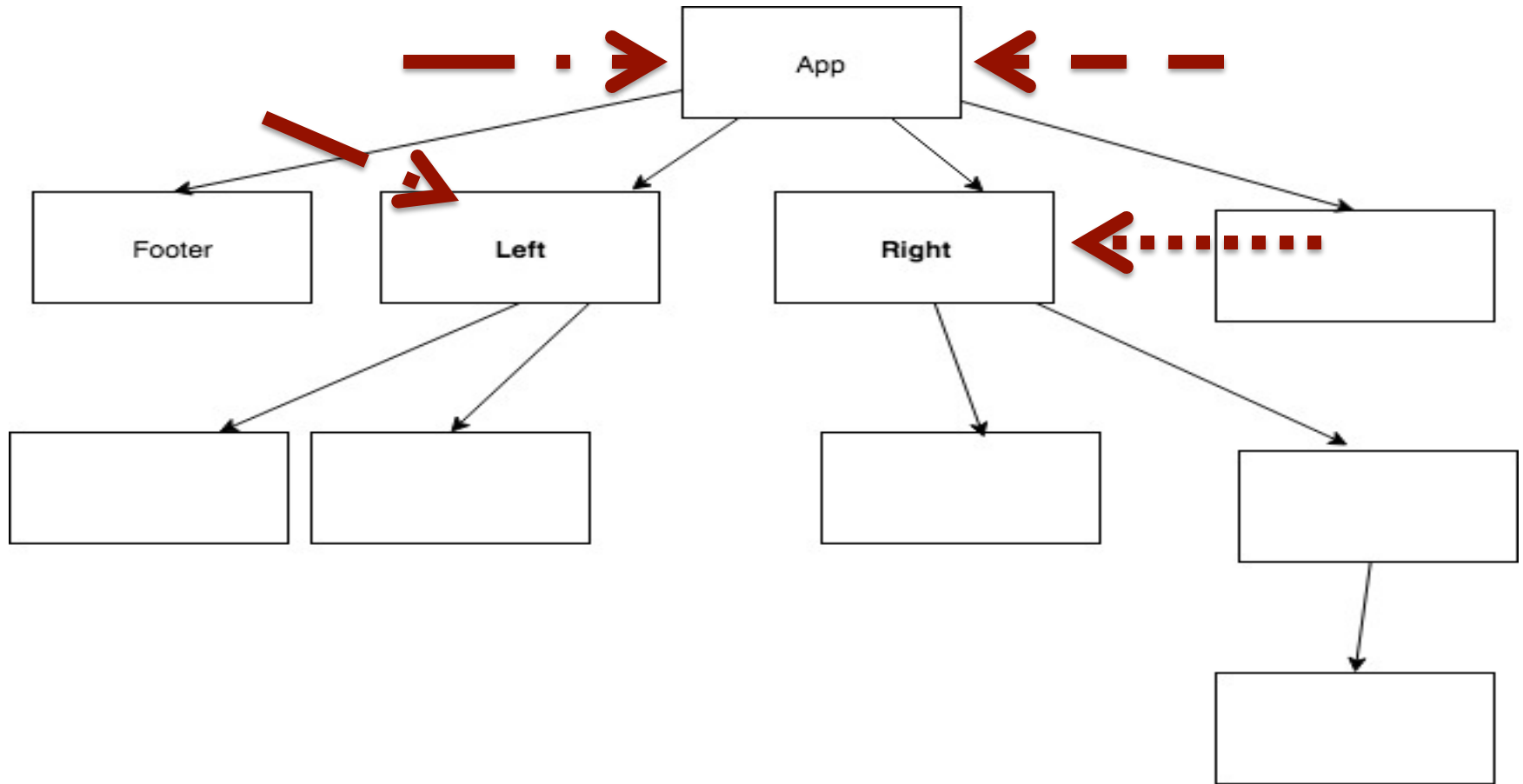
## Step 4: Identify where state should live.



# Abstract example



# Where State lives - Abstract example



# Sample: Filtered Friends app

- **Only 1 state variable (searchText).**
- **Design A – 1-way data flow design.**
  - FriendsApp component needs to display the text & use it to compute the filtered list of friends.
  - No other component uses this state.
- **Design B – Inverse Data Flow design.**
  - FriendsApp component needs it to compute the filtered list of friends.
  - SearchBox component needs to display it in the text field.
  - FriendsApp is a ‘common’ component
- **Can now add state initialization code to ststeful components.**

## Step 5: Add inverse data flow

- **Problem: A component's state changes when the user interaction with a deeper nested component.**
  - **The browser event occurs in the nested component.**
  - **The nested component must communicate the event to the (superordinate) stateful component.**
- **Solution: Inverse data flow pattern:**
  - **Stateful component passes (as a prop) a local function reference to the nested component.**
  - **Nested component calls function when event fires.**
- **Update Storybook stories to reflect additional props**

# Developing a React web app

- **Step 1: Break the UI into a component hierarchy.**
- **Step 2: Build a static version in React.**
- **Step 3: Identify the minimal representation of UI state.**
- **Step 4: Identify where your state should live.**
- **Step 5: Add inverse data flow, if required.**

